Schema Meta-Matching

Extended abstract

Carmel Domshlak DCARMEL@CS.CORNELL.EDU
Dept. of Computer Science, Cornell University, Ithaca, NY 14853

Avigdor Gal AVIGAL@IE.TECHNION.AC.IL
Technion – Israel Institute of Technology, Technion City, Haifa 32000, Israel

1. Introduction

Schema matching, the process of matching between concepts describing the meaning of data in heterogeneous, distributed data sources (*e.g.* database schemata, XML DTDs, HTML form tags, *etc.*) is one of the basic operations required by the process of data integration. Recently, several algorithms for automatic schema matching have been proposed and evaluated in the database community. While in many domains these tools succeed in finding the right matching, empirical analysis shows that there is not (and probably will never be) any single algorithm that is guaranteed to succeed in all possible domains and applications. To overcome this problem, several tools are being developed that combine the principles by which different algorithms judge the similarity between concepts. In parallel, Anaby-Tavor *et al* [1] takes another approach, by which not one, but K best-ranked mappings are generated, then examined iteratively until a good mapping is found.

In this paper we introduce a novel framework for schema matching which we call *schema meta-matching*. This approach extends the idea of working with top-K mappings, applying it to an arbitrary ensemble of algorithms for schema matching. Informally, schema meta-matching is the problem of computing a "consensus" ranking of alternative mappings between two sets of concepts, given the "individual" graded rankings provided by *several* algorithms for schema matching.

We begin with an overall look at schema matching, concluding with a discussion of how the field is likely to develop in the near future. We formalize the problem of schema meta-matching and introduce several algorithmic solutions for this problem, including one that adapts standard techniques for general quantitative rank aggregation, and others employing novel techniques specific to the problem of schema matching. We provide a formal analysis of the applicability and relative performance of each competing algorithm. Finally, we show how combining these approaches results in the most successful matchings.

1.1 Motivation

Due to the cognitive complexity of matching between sets of concepts [2], this task has traditionally been performed by human experts [7]. As the process of data integration has become more automated, the ambiguity inherent in concept interpretation, also known as semantic heterogeneity, has become one of the main obstacles to this process. For obvious reasons, manual concept reconciliation in dynamic environments (with or without computer-aided tools) is inefficient and at times close to impossible. Introduction of the Semantic Web vision and shifts toward machine-understandable Web resources have made even clearer the vital need for automatic matching between sets of concepts, also known as *schema matching* [10].

Several tools for automated schema matching, such as Cupid [8] and OntoBuilder [9], have been developed in recent years. Given two data schemata (*i.e.* two sets of concepts) S and

S', these tools output a single *mapping* from the elements of S to the elements of S'. This mapping is considered by the tool to be the "best mapping" among all possible mappings from S to S'. However, experience with such tools [6] makes clear that automatic matching carries a degree of uncertainty, due to the ambiguity and heterogeneity (both syntactic and semantic) of the data description concepts. It is simply unrealistic to expect any single mapping engine to provide a correct matching for any possible schemata pair.

Cupid [8] and OntoBuilder [9], as well as other models for automatic schema matching, have sought to address this problem by combining different matching techniques, employing a weighted sum of alternative (possibly not entirely independent) similarity measures to specify similarity between pairs of concepts in S and S'. An approach different in principle (and tangential to the choice of basic/compound similarity measure) is proposed in [1]. Here, instead of a single mapping, K best mappings are generated and examined iteratively until a good mapping is found. This approach is significantly more flexible than the standard approach, in that the "exact" mapping (*i.e.* the mapping that would have been chosen by the proper human expert) may now be ranked lower than the "best" mapping, yet still be identified during the automated matching process. However, in order to keep the matching process efficient, the exact mapping must still appear within the best K mappings for some reasonably small K.

1.2 Challenges

On the positive side, an empirical analysis of the top-K approach in [1] demonstrates that, for a certain significant class of mapping algorithms (referred to in [6] as "monotonic"), the required K is expected to be substantially smaller than the size of the search space. This makes the top-K approach an attractive alternative to the standard (*i.e.* top-1) methods. However, the same empirical analysis also confirms that *there is no single dominant algorithm for determining similarity between different concepts that performs best, regardless of the data model and application domain.* Therefore, especially with the expected growth of the Semantic Web, it is unavoidable that every algorithm will perform badly in some domains, even if in other domains it will be unbeatable.

To summarize our perspective on the present and future of schema matching, we expect the growth of the Semantic Web and rising interest in the area of data integration to result in the development of numerous tools for schema matching that will be freely accessible on the Web. Second, due to the enormous heterogeneity and ambiguity of data descriptions, we believe that the algorithms for schema matching will be based on a rich variety of (possibly conceptually different) techniques. Finally, specific algorithms already tend to be "experts" in particular realms, and with time it will become even more difficult to automatically characterize a priori domains in which a given algorithm will work better than others.

Bearing these observations in mind, we believe that customers of schema matching have the right to expect some degree of robustness, despite the biases and shortcomings of individual algorithms. For this, we need robust and efficient aggregation techniques for schema *meta-matching*.

In the ideal scenario for rank aggregation, each judge (or schema-matching algorithm, in our case) ranks all possible cases in the universe of alternatives, where each alternative is associated with a level of "goodness" as set by the judge. Unfortunately, in the case of schema matching, the size of the universe of alternatives makes this unrealistic: Given two schemata, each consisting of n concepts, there are $n!$ alternative 1:1 mappings between them. Therefore, any method for schema meta-matching will have to either consider individual rankings represented implicitly in some compact form, or carefully query the judges about the mappings, limiting the number and complexity of these queries to the extent possible. Note that in contrast to the case of the Web meta-search [3], our judges are willing to answer any query

about mapping rankings. Of course, we should distinguish between different types of queries, as these can differ dramatically in complexity from the perspective of both time and space.

1.3 Our results

We begin by formalizing the framework of schema meta-matching. We extend the idea of generating top-K mappings [1], applying it to an arbitrary ensemble of algorithms for schema matching. In particular, our formalization of the problem does not require explicit listing of the individual rankings a priori, preserving the output of the individual schema matching algorithms in a compact form of similarity matrices.

Our schema meta-matching has been inspired both by the needs of the data integration process, and by some recent works on rank aggregation techniques in the areas of Web search and database middleware [3, 4]. First, we show that the Threshold algorithm, originally proposed in the context of database middleware [4], can be applied to our problem almost as is. Unfortunately, as we show, computing top-K mappings for schema meta-matching using the Threshold algorithm can take exponentially more time than the size of the matched schemata. Since in its original context, the Threshold algorithm has been shown to be optimal in a strong sense, we develop techniques that exploit both the specifics of schema matching and the properties of the TKM (Top-K Mapping) algorithm for generating K best mappings between two schemata (with respect to a single judge) [1]. For a certain wide class of problems, we present a simple algorithm, called Matrix-Direct, the time complexity of which is polynomial in terms of the size of the matched schemata and the required K. Subsequently, we present the Matrix-Direct with Bounding algorithm, which draws upon both Matrix-Direct and Threshold algorithms, addressing the problems where Matrix-Direct is inapplicable, while being significantly more efficient than the Threshold algorithm in at least some schema meta-matching problems.

We show that the Threshold and Matrix-Direct with Bounding algorithms are (performance-wise) mutually undominated–that is, there exist problem instances in which one algorithm performs dramatically better than the other. Therefore, we introduce a hybrid version of the two algorithms, based on their in-parallel, mutually-enhancing execution. Our analysis shows the complexity implications of such algorithm hybridizing.

2. Schema Matching: Background and Notation

The standard process of schema matching has two steps. First, given two sets of concepts S and S' (henceforth referred to as *schemata*), a real-valued degree of similarity is automatically computed for all possible pairs of concepts from $S \times S'$. In what follows, we assume that S and S' are of identical arity, *i.e.* there are n attributes in each schema. Since in this paper we restrict ourselves to 1:1 mappings, this assumption causes no loss of generality, as a smaller schema can always be extended by means of dummy concepts. Therefore, the first step results in an $n \times n$ *similarity matrix* M, where $M_{i,j}$ represents the degree of similarity between the i-th attribute of S and j-th attribute of S'. Note that algorithms for schema matching differ mainly in the measures of similarity that they employ. These measures can be arbitrarily complex or typically procedural, and may use various techniques for name matching, domain matching, structure matching (such as XML hierarchical representation), *etc.*

In the second step of the process, the similarity information in M is used to quantify the quality of different mappings from the concepts in S to the concepts in S'. Since the matching algorithms are assumed to be symmetric, we denote a 1-1 mapping from S and S' as a permutation σ of $1, \ldots, n$, meaning that the i-th concept of S is mapped to the $\sigma(i)$-th concept of S'. A single mapping from S to S' is then chosen by the algorithm to be the *best mapping*, typically the one that maximizes some *local aggregation function* (or *l-aggregator*, for

short)

$$f(\sigma, M) = f(M_{1,\sigma(1)}, \ldots, M_{n,\sigma(n)})$$

i.e., a function that aggregates the degrees of similarity associated with the pairs of concepts forming the mapping. Probably the most popular choice of l-aggregator is the sum (or average) of the pair-wise degrees of similarity of the mapped concepts. Without loss of generality, in what follows we assume that f is computable in time linear in n. However, at least technically, nothing prevents us from using more sophisticated l-aggregators.

The major shortcoming of standard methods for schema matching is that they commit to the "best mapping." The problem is that, due to ambiguity in concept interpretation, the "best mapping" chosen by the algorithm can actually be an unsuccessful choice. A more flexible approach has been proposed in [1]: Instead of generating just the"best" mapping, a set of top-K mappings are generated and examined iteratively until a good mapping is found.[1] In this approach, the "exact mapping" is likely to be identified if the algorithm ranks it sufficiently high (but not necessarily as the best).

Let Σ denote the set of $n!$ possible mappings from S to S'. The i-th best mapping $\sigma^i \in \Sigma$ is defined recursively as:

$$\sigma^i = \underset{\sigma}{\operatorname{argmax}} \left\{ f(\sigma, M) \mid \sigma \in \Sigma \setminus \{\sigma^1, \cdots, \sigma^{i-1}\} \right\} \tag{1}$$

where M is the similarity matrix determined by the algorithm, and f is the l-aggregator in use. One of the key contributions of [1] is an efficient algorithm for generating top-K mappings between a pair of schemata. In what follows, we refer to this algorithm as TKM (short for "top-K mappings"). This algorithm is generic in the sense that it can embed as a subroutine any similarity measure between a pair of concepts, and can use most of the popular operators for scoring the weight of a schema-to-schema mapping. The time complexity of TKM is $O(Kn \cdot \Phi)$, where Φ is the time complexity of the best (complexity-wise) algorithm for finding a maximum weight mapping in a bipartite graph. To the best of our knowledge, today we have $\Phi = O(n^3)$ [5], and so the time complexity of TKM is $O(Kn^4)$. The space complexity of TKM is $O(nK)$. For a detailed description of the TKM algorithm, we refer the interested reader to [1].

3. Rank Aggregation for Schema Mappings

That the idea of exploiting an ensemble of algorithms for schema matching has emerged is not surprising, especially given that this idea has already been discussed in recent works on data integration [8, 9]. Informally, we consider such a set of algorithms as a group of experts that may differ in their judgment on the degree of similarity between various concepts. Our goal is to aggregate the alternative opinions in the group, ultimately reaching a single position regarding the relative goodness of different mappings. In what follows, we refer to this problem as *schema meta-matching*.

3.1 Problem Statement

Consider a set of m algorithms for schema matching $A_1, \ldots, A_m$. Given two schemata S and S' as before, these algorithms produce $n \times n$ similarity matrices $M^{(1)}, \ldots, M^{(m)}$, respectively, where $M^{(l)}_{i,j}$ represents the degree of similarity that the expert A_l associates with mapping the i-th attribute of S to the j-th attribute of S'. In what follows we assume that, while differing in

1. What constitutes as a good mapping is beyond the scope of this paper. Suffice it to say that the process of mappings evaluation is tool dependent, and may involve the analysis of query variations, analysis of Web server error messages, *etc.*

their opinions on the similarity between concepts, all m algorithms use the same l-aggregator f.

Given such a set of algorithms $A_1, \ldots, A_m$, we would like to aggregate the weights provided by the algorithms to the mappings, and to reason about the resulting aggregated ranking of alternative mappings. Such an aggregation can be modeled using a *global aggregation function* (or *g-aggregator*, for short) $F\left(f(\sigma, M^{(1)}), \cdots, f(\sigma, M^{(m)})\right)$. For instance, a natural candidate for g-aggregator is a (weighted) sum or average of the local rankings. For ease of presentation and without loss of generality, we assume that F is computable in time linear in m. In what follows, we denote such a pair of aggregators by $[\![f, F]\!]$, where the first and the second elements of this ordered tuple stand for the l-aggregator and g-aggregator, respectively. Likewise, we use the notation $[\![f, F]\!](\sigma) \equiv F\left(f(\sigma, M^{(1)}), \cdots, f(\sigma, M^{(m)})\right)$ for the aggregated value assigned by $[\![f, F]\!]$ to the mapping σ.

The formal problem that we consider here is that of generating top-K mappings from S to S' with respect to a set of algorithms $A_1, \ldots, A_m$ and a global aggregation function F. This problem is well defined, and the i-th best mapping $\sigma^i \in \Sigma$ is defined recursively as:

$$\sigma^i = \operatorname*{argmax}_{\sigma} \left\{ [\![f, F]\!](\sigma) \mid \sigma \in \Sigma \setminus \{\sigma^1, \cdots, \sigma^{i-1}\} \right\}, \tag{2}$$

much as it is defined in Eq. 1 for the basic case of $m = 1$.

In the remainder of this paper we focus on the algorithmic aspects of solving this problem. Before we discuss the algorithms themselves, it is worth observing that a naïve approach of (i) generating m top-K lists with respect to the algorithms $A_1, \ldots, A_m$ using TKM, and (ii) subsequent aggregation of these lists using F is not sound. To illustrate, consider the mapping σ^1, *i.e.*, the best mapping with respect to Eq. 2. First, strange as it may seem, σ^1 may appear in none of the m individual top-K lists, and thus will definitely not appear in an aggregated list of any length. Second, even if σ^1 appears in some, or even most, individual top-K lists, it can be improperly ranked in step (ii), or even discarded from the aggregated top-K list. Therefore, though intuitively TKM would seem to be an essential tool for any schema matching problem, "querying" the algorithms independently using only TKM does not resolve the question at hand.

3.2 The Threshold Algorithm

The problem of how to achieve optimal aggregation of several quantitatively ordered lists is not new, and has recently been studied extensively in the context of middleware for multimedia database systems [4]. The most efficient general algorithm for this problem, called the Threshold algorithm (TA, for short), has been introduced in [4], and we begin by presenting this algorithm in terms of our problem in Figure 1.

The intuition behind TA is elegantly simple. Assume that $K = 1$, *i.e.* we are interested only in the best mapping. Assume that we are at a stage in the algorithm where we have not seen any mapping σ whose aggregated weight $[\![f, F]\!](\sigma) \geq \tau_{TA}$. Therefore, we cannot be sure at this point that the best mapping has already been seen, since the next mapping σ' generated by TKM could have aggregated weight $[\![f, F]\!](\sigma') = \tau_{TA}$. If this is the case, then clearly no mapping seen so far is the best mapping, since $[\![f, F]\!](\sigma') > [\![f, F]\!](\sigma)$. Only when we see a mapping whose aggregated weight is at least τ_{TA} is it safe to halt. Similarly, for $K > 1$, the stopping rule verifies a sufficient condition to ensure that the top K mappings have been seen.

The only property required to ensure the completeness of TA is monotonicity of the g-aggregator F in the following sense [4]: A function F is called *monotone* if, for every two mappings σ, σ' such that $f(\sigma, M^{(l)}) > f(\sigma', M^{(l)})$ holds for all $1 \leq l \leq m$, we have $[\![f, F]\!](\sigma) > [\![f, F]\!](\sigma')$. This requirement does not seem to induce any practical limitation. Therefore, henceforth we adopt this assumption of monotonicity for g-aggregators.

Algorithm TA

1. Run $A_1, \ldots, A_m$, generating the similarity matrices $M^{(1)}, \ldots, M^{(m)}$.
2. Do incremental, parallel evaluations of TKM on $M^{(1)}, \ldots, M^{(m)}$. These evaluations are unbounded, and they correspond to a sorted access in parallel to each of the m sorted lists of all $n!$ alternative mappings.
   (a) As a mapping σ is seen for the first time in one of these sorted lists, compute the remaining $f(\sigma, M^{(1)}), \ldots, f(\sigma, M^{(m)})$, and the aggregated weight $[\![f, F]\!](\sigma)$. If this weight is one of the K highest we have seen so far, then remember σ.
   (b) For each $M^{(l)}$, let σ_l be the last mapping generated by TKM. Define the threshold value $\tau_{TA} = F\left(f(\sigma_1, M^{(1)}), \cdots, f(\sigma_m, M^{(m)})\right)$. If at least K mappings have been seen whose weight is at least τ_{TA}, then halt.
3. Let Y be a set containing K mappings with the highest grades seen so far. The output is then the graded set $\{[\sigma, [\![f, F]\!](\sigma)] \mid \sigma \in Y\}$.

Figure 1: The Threshold Algorithm (TA), adopted for schema meta-matching.

The next theorem, which follows immediately from the definition of TA, describes the space complexity of TA when it is applied in our domain. (The proofs of all formal claims in this paper are deferred to the appendix.)

Theorem 1 *Space requirements of* TA *for schema meta-matching consist of* m TKM *buffers, and a single additional buffer whose size is* $O(nK + m)$.

Recall from Section 2 that the size of each TKM buffer is $O(nK')$, where K' is the actual number of the TKM iterations required by TA. The problem is that K' depends on the suitability of the algorithms $A_1, \cdots, A_m$ for the domain of the schemata in question. Unfortunately, the next theorem shows that TA for schema meta-matching may have exponentially long runs. [2]

Theorem 2 *The time complexity of* TA *for schema meta-matching is* $\Omega((\frac{n}{2})!)$.

3.3 The Matrix-Direct Algorithm

Theorem 2 provides a strong motivation to seek more efficient alternatives to the TA algorithm. However, in [4], TA is shown to be optimal in a strong sense, namely "instance optimal." For the formal definition of instance optimality we refer the reader to [4]. Very roughly, for any set of data and any other rank aggregation algorithm A, the time complexity of $Comp(\mathsf{TA}) = O(Comp(A))$. Hence, at least at first glance, it seems that using TA for schema meta-matching is the best we can do. Fortunately, exploiting specifics of the schema matching problem allows us in many cases to perform significantly better using an extremely simple technique. (Note that this does not contradict the instance optimality of TA, as TA is a generic algorithm, independent of the actual grading mechanisms.)

Let us consider what is probably the most conservative principle for schema mappings rank aggregation, namely:

$$f(\sigma, M^{(l)}) = \sum_{i=1}^{n} M^{(l)}_{i,\sigma(i)} \qquad [\![f, F]\!](\sigma) = \sum_{l=1}^{m} k_l f(\sigma, M^{(l)}) \tag{3}$$

2. Ignoring the fact that the sorted lists of objects (= mappings) in our application are generated by TKM with respect to a set of similarity matrices, it can be easily shown that TA may have to access in a sorted manner as many as half of each sorted list (*e.g.*, see Example 6.3 in [4]).

Algorithm MD

1. Run $A_1, \ldots, A_m$, generating $M^{(1)}, \ldots, M^{(m)}$.
2. Construct a new matrix M^*, where, for $1 \leq i, j \leq n$, $M^*_{i,j} = F(M^{(1)}_{i,j}, \cdots, M^{(m)}_{i,j})$.
3. Using TKM, generate top-K mappings with respect to M^* and the l-aggregator function $f(\sigma, M^*)$.

Figure 2: The Matrix-Direct (MD) Algorithm.

Notice that the order of summation in Eq. 3 can be exchanged, resulting in $[\![f, F]\!](\sigma) = \sum_{i=1}^{n} \sum_{l=1}^{m} k_l M^{(l)}_{i,\sigma(i)}$. The special case of Eq. 3 can be generalized as follows. Given a pair of l-aggregator f and g-aggregator F, we say that f and F are *mutually commutative* if and only if, for every set of similarity matrices $M^{(1)}, \cdots, M^{(m)}$ and for every mapping σ, we have:

$$[\![f, F]\!](\sigma) = [\![F, f]\!](\sigma) \tag{4}$$

For such pairs of l- and g-aggregators, in Figure 2 we present the Matrix-Direct algorithm (or MD, for short).

Theorem 3 *Given a set of m algorithms for schema matching, and a pair of mutually commutative local and global aggregation functions $[\![f, F]\!]$, the* MD *algorithm correctly finds top-K mappings with respect to the aggregative ranking in time $O(n^2 m + C(\mathsf{TKM}))$, where $C(\mathsf{TKM})$ denotes the time complexity of the TKM algorithm.*

The next theorem, which follows immediately from the definition of MD, describes the space complexity of MD, further distinguishing MD from TA.

Theorem 4 *Space requirements of MD consist of only a single TKM buffer.*

4. Matrix-Direct Algorithm with Bounding

Reading so far, it seems we can conclude that given a schema meta-matching problem, if the l- and g-aggregators f and F are mutually commutative, then proceed with MD. Otherwise, proceed with TA (*i.e.*, we are back to a general, instance optimal algorithm for quantitative rank aggregation). However, below we show that, while the former conclusion is sound, the latter is not necessarily so.

Consider a pair of similarity matrices S and S', and two pairs of l- and g-aggregators $[\![f, F]\!]$ and $[\![f', F']\!]$. We say that $[\![f, F]\!]$ is *dominated by* $[\![f', F']\!]$ on S, S'(denoted as $[\![f, F]\!] \prec [\![f', F']\!]$) if, for every mapping $\sigma \in \Sigma$, we have:

$$[\![f', F']\!](\sigma) \geq [\![f, F]\!](\sigma) \tag{5}$$

Now, suppose that we are given a pair of l- and g-aggregators $[\![f, F]\!]$ that are not mutually commutative, for which there exists another pair of functions $[\![h, H]\!]$ that *are* mutually commutative, and we have $[\![f, F]\!] \prec [\![h, H]\!]$. For example, let F be a weighted sum as in Eq. 3, and f be defined as:

$$f(\sigma, M) = \begin{cases} \sum_{i=1}^{n} M_{i,\sigma(i)}, & \sum_{i=1}^{n} M_{i,\sigma(i)} > t \\ 0, & \text{otherwise} \end{cases}$$

where $t > 0$ is some predefined constant threshold. This example reflects a possible setting in schema matching [9], in which individual rankings that do not pass a user defined threshold are nullified. It is not hard to verify that f and F are not mutually commutative. On the

other hand, functions h and H standing for simple sum and weighted sum (as in Eq. 3) are mutually commutative, and we have $[\![f, F]\!] \prec [\![h, H]\!]$.

For such cases we now present the Matrix-Direct algorithm with Bounding (or MDB, for short). This algorithm draws upon both TA and MD, addressing problems for which MD is incomplete (namely, problems with non-commutative pairs of local and global aggregation functions), while being more efficient than TA in at least some such problem instances.

Algorithm MDB

1. Run $M_1, \ldots, M_m$, generating $M^{(1)}, \ldots, M^{(m)}$. Generate a matrix M^*, where each entry $M^*_{i,j} = H(M^{(1)}_{i,j}, \cdots, M^{(m)}_{i,j})$.
2. Do iterative parallel evaluation of TKM on M^* using l-aggregator h.
   (a) As a mapping σ is generated, compute the actual aggregated weight $[\![f, F]\!](\sigma)$. If this weight is one of the K highest we have seen so far, then remember σ.
   (b) Define the *MD threshold value* τ_{MD} to be $h(\sigma, M^*)$.[a] As soon as at least K mappings have been seen whose weight is at least τ_{MD}, then halt the loop.
3. (Similarly to TA) Let Y be a set containing K mappings with the highest grades seen so far. The output is then the graded set $\{[\sigma, [\![f, F]\!](\sigma)] \mid \sigma \in Y\}$.

a. It is worth noting that, due to mutual commutativity of h and H, we have $\tau_{MD} = [\![H, h]\!](\sigma) = [\![h, H]\!](\sigma)$.

Figure 3: The Matrix-Direct with Bounding (MDB) algorithm.

The MDB algorithm is shown in Figure 3. Given a schema meta-matching problem with non-commutative aggregators $[\![f, F]\!]$, the idea of MDB is to use a dominating pair of mutually commutative functions $[\![h, H]\!]$ as an upper bound for the "uncomfortable" $[\![f, F]\!]$ that are of actual interest. Informally, MDB behaves similarly to MD if the latter is given with the aggregators $[\![h, H]\!]$. However, instead of reporting immediately on the generated mappings σ, MDB uses the decreasing aggregated weights $[\![h, H]\!](\sigma)$ to update the value of a threshold τ_{MD}. In turn, much as the way the threshold τ_{TA} is used in the TA algorithm, the threshold τ_{MD} is used to judge our progress with respect to the weights $[\![f, F]\!]$ that really matter.

Theorem 5 shows that MDB is correct for any such upper bound $[\![h, H]\!]$. Of course, the performance of MDB depends crucially on the quality of the bounding, *i.e.*, on the tightness of $[\![h, H]\!]$ as an upper bound for $[\![f, F]\!]$.

Theorem 5 *Given a set of m algorithms for schema mappings, a pair of local and global aggregation functions $[\![f, F]\!]$, and a pair of mutually commutative functions $[\![h, H]\!]$ such that $[\![f, F]\!] \prec [\![h, H]\!]$, the MDB algorithm correctly finds top-K mappings with respect to $[\![f, F]\!]$.*

Returning to the question of performance, recall that our intention in developing MDB was to provide an alternative to TA for those cases where the standard MD is not applicable. Have we achieved our goal, or will TA always be preferable anyway? We now show that, for schema meta-matching, TA is no longer instance optimal, since MDB can outperform TA. Furthermore, we show that in some instances, TA can be *significantly* worse than MDB.

Theorem 6 *Given a schema meta-matching problem instance, time complexity of TA on this instance can be exponentially worse than that of MDB.*

Algorithm Hybrid

1. Run $M_1, \ldots, M_m$, generating $M^{(1)}, \ldots, M^{(m)}$. Generate a matrix M^*, where each entry $M^*_{i,j} = H(M^{(1)}_{i,j}, \cdots, M^{(m)}_{i,j})$.
2. Perform incremental, parallel evaluation of TKM with f on $M^{(1)}, \ldots, M^{(m)}$, and with h on M^*.
   (a) As a mapping σ is seen for the first time in one of these sorted lists, generate the remaining $f(\sigma, M^{(1)}), \ldots, f(\sigma, M^{(m)})$, and compute the aggregated weight $[\![f, F]\!](\sigma)$. If this weight is one of the K highest we have seen so far, then remember σ.
   (b) For each $M^{(1)}, \ldots, M^{(m)}, M^*$, let $\sigma_1, \ldots, \sigma_m, \sigma_*$ be the last mappings generated by TKM, respectively. Define the threshold value $\tau = \min(\tau_{TA}, \tau_{MDB})$, where $\tau_{TA} = F\Big(f(\sigma_1, M^{(1)}), \cdots, f(\sigma_m, M^{(m)})\Big)$ and $\tau_{MDB} = h(\sigma_*, M^*)$. As soon as at least K mappings have been seen whose weight is at least τ, then halt the loop.
3. (Similarly to TA) Let Y be a set containing K mappings with the highest grades seen so far. The output is then the graded set $\{[\sigma, [\![f, F]\!](\sigma)] \mid \sigma \in Y\}$.

Figure 4: The Hybrid algorithm, combining TA and MDB.

5. Merging TA and MDB

Although Theorem 6 shows that TA does not dominate MDB, we show that TA is also not dominated by MDB. Furthermore, the time complexity of MDB can be significantly greater than that of TA.

Theorem 7 *Given a schema meta-matching problem instance, time complexity of MDB on this instance can be exponentially worse than this of TA.*

The main conclusion to be drawn from Theorem 7 is that MDB should not replace, but rather should complement, the standard TA algorithm. In Figure 4 we present an algorithm that combines both TA and MDB, referring to this algorithm as Hybrid.

Informally, running Hybrid constitutes a parallel execution of TA and MDB, i.e. performing $m + 1$ parallel executions of the TKM procedure. The embedded evaluations of TA and MDB are not independent, but communicating and mutually enhancing. The basic idea of Hybrid is to aggregate the thresholds used in TA and MDB, achieving a new threshold that is better than the original two. The hybrid threshold $\tau = \min(\tau_{TA}, \tau_{MDB})$ is not redundant with respect to τ_{TA} and τ_{MDB}, i.e. both cases $\tau < \tau_{TA}$ and $\tau < \tau_{MDB}$ are feasible. Therefore, the schema matchings selected by TA as candidates for the top-K set can be "approved" by means of the information obtained through MDB, and vice versa. Finally, theorem 8 below states that this way of mixing the thresholds preserves the correctness of the matchings achieved. Notice that, from the perspective of time complexity, theorems 6 and 7 show that employing TA and MDB together makes sense. Likewise, theorems 1 and 4 show that running MDB in addition to TA results in only a slight increase in space complexity.

Theorem 8 *Given a set of m algorithms for schema mappings, a pair of local and global aggregation functions $[\![f, F]\!]$, and a pair of mutually commutative functions $[\![h, H]\!]$ such that $[\![f, F]\!] \prec [\![h, H]\!]$, the Hybrid algorithm correctly finds top-K mappings with respect to $[\![f, F]\!]$.*

6. Conclusions and Directions for Future Work

We have introduced schema meta-matching, a novel framework for robust schema matching that exploits an ensemble of algorithms for schema matching. We have explored algorithmic and computational aspects of this framework, using techniques for quantitative rank

aggregation developed in the area of database middleware, as well as novel techniques that we have developed especially for the problem of schema matching. In particular, we have presented a formal comparative computational analysis of alternative algorithms for schema meta-matching. We are currently working on implementing the proposed algorithms, and conducting a set of comparative experiments, using real-life schema data gathered from Web forms and an ensemble of state-of-the-art algorithms for schema matching.

Our work opens several directions for future research, and we are currently exploiting some of them. First, in this paper we focused only on 1-1 schema matchings between pairs of schemata. However, the area of data integration is also struggling with less committing types of matching, such as n-1 and n-m. While our schema meta-matching framework covers all types of matching, the efficiency and even suitability of the specific algorithms discussed in this paper still require a thorough analysis in the presence of alternative matching goals.

Second, the complexity of our MDB algorithm depends crucially on the quality of the chosen pair of dominating aggregators. Therefore, it will be helpful to refine the notion of dominance by incorporating measures of the actual tightness of dominance. Third, the existing methods for incremental aggregation of quantitative rankings (and our algorithms are no exception) advance over all individual rankings in question uniformly. We believe it is worthwhile investigating the potential benefits of advancing non-uniformly on individual lists, reducing the overall complexity of rank aggregation by exploiting specifics of the actual problem domain.

Acknowledgments

We thank Ilan Shimshoni for useful discussions. The work of Gal was partially supported by Technion V.P.R. Fund – E. and J. Bishop Research Fund, the Fund for the Promotion of Research at the Technion, and the IBM Faculty Award for 2003/2004 on "Self-Configuration in Autonomic Computing using Knowledge Management." The work of Domshlak was partially supported by the Intelligent Information Systems Institute, Cornell University (AFOSR grant F49620-01-1-0076). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

References

[1] A. Anaby-Tavor, A. Gal, and A. Moss. Efficient algorithms for top-k matchings. Submitted for publication. Available upon request from avigal@ie.technion.ac.il, 2003.

[2] B. Convent. Unsolvable problems related to the view integration approach. In *ICDT-86*, 1986. In *Computer Science*, Vol. 243, pp. 141-156.

[3] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW-01*, pages 613–622, 2001.

[4] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66:614–656, 2003.

[5] M. L. Freedman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *FOCS-84*, pages 338–346, 1984.

[6] A. Gal, A. Anaby-Tavor, A. Trombetta, and D. Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *VLDB Journal*, 2004. to appear.

[7] R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *PODS-97*, pages 51–61, 1997.

[8] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *VLDB-01*, pages 49–58, 2001.

[9] G. Modica, A. Gal, and H. Jamil. The use of machine-generated ontologies in dynamic information seeking. In *CoopIS-01*, pages 433–448, 2001.

[10] E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

Appendix A. Proofs

Theorem 1 Space requirements of TA for schema meta-matching consist of m TKM buffers, and a single additional buffer whose size is $O(nK + m)$.

Proof: The requirement for m TKM buffers is apparent, and the last (n-independent) buffer is consumed by TA itself. As it is shown in [4], all that TA must remember is the current top K mappings, and (pointers to) the last mappings generated by TKM for each one of the m algorithms. ∎

Theorem 2 The time complexity of TA for schema meta-matching is $\Omega((\frac{n}{2})!)$.

Proof: Although more realistic examples are possible, for ease of presentation, we use a synthetic example that is simple to present. Consider two algorithms, A_1 and A_2, and a pair of schemata S and S', each consisting of n concepts, where $n = 2k$, $k \in \mathbb{N}$. Likewise, let the l-aggregator f be the regular *product*, and the g-aggregator F be the utilitarian aggregator *min*.

Given S and S', the similarity matrices $M^{(1)}$ and $M^{(2)}$, induced by A_1 and A_2, respectively are as follows:

$$M^{(1)}_{i,j} = \begin{cases} x, & (i \leq n/2) \wedge (j \leq n/2) \wedge (i \neq j) \\ x - \epsilon, & i = j \\ 0, & \text{otherwise} \end{cases} \qquad M^{(2)}_{i,j} = \begin{cases} x, & i > n/2 \wedge j > n/2 \wedge i \neq j \\ x - \epsilon, & i = j \\ 0, & \text{otherwise} \end{cases}$$

for arbitrary values of x and ϵ, where $\epsilon \ll x$, and $x - \epsilon > 0$. Below we illustrate such matrices for $n = 4$:

$$M^{(1)} = \begin{pmatrix} x-\epsilon & x & 0 & 0 \\ x & x-\epsilon & 0 & 0 \\ 0 & 0 & x-\epsilon & 0 \\ 0 & 0 & 0 & x-\epsilon \end{pmatrix} \qquad M^{(2)} = \begin{pmatrix} x-\epsilon & 0 & 0 & 0 \\ 0 & x-\epsilon & 0 & 0 \\ 0 & 0 & x-\epsilon & x \\ 0 & 0 & x & x-\epsilon \end{pmatrix}$$

First, consider $M^{(1)}$. Each mapping between the first $n/2$ concepts of S and the first $n/2$ concepts of S' (see the top left quadrant of $M^{(1)}$) results in a non-zero value of f restricted to these concepts. There are $(\frac{n}{2})!$ such mappings. Any other mapping of any of these concepts will nullify the value of f. On the other hand, the last $n/2$ concepts of S have to be mapped to the $n/2$ last concepts of S', and there is only one such mapping leading to a non-zero value of f, namely the main diagonal of the bottom right quadrant of $M^{(1)}$. Therefore, we have constructively shown that $M^{(1)}$ induces exactly $(\frac{n}{2})!$ mappings σ such that $f(\sigma, M^{(1)}) > 0$. Denote this set of mappings as Σ^+_1. By a similar construction one has that the same holds for $M^{(2)}$, *i.e.*, $|\Sigma^+_2| = (\frac{n}{2})!$.

Now consider the sets Σ^+_1 and Σ^+_2, and let σ_I denote the indentity mapping, i.e. the mapping captured by the main diagonals of $M^{(1)}$ and $M^{(2)}$. Evidently, for $i \in \{1, 2\}$, we have $\sigma_I \in \Sigma^+_i$, and, for each $\sigma_I \neq \sigma \in \Sigma^+_i$, we have $f(\sigma, M^{(i)}) > f(\sigma_I, M^{(i)})$. Therefore, TKM on both $M^{(1)}$ and $M^{(2)}$ will reach σ_I after exactly $(\frac{n}{2})!$ iterations. On the other hand, we have $\Sigma^+_1 \cap \Sigma^+_1 = \{\sigma_I\}$, and thus, for each mapping $\sigma \in \Sigma$, we have:

$$[\![f, F]\!](\sigma) = \min \left\{ \prod_{i=1}^{n} M^{(1)}_{i,\sigma(i)}, \prod_{i=1}^{n} M^{(2)}_{i,\sigma(i)} \right\} = \begin{cases} n(x - \epsilon), & \sigma = \sigma_I \\ 0, & \text{otherwise} \end{cases}$$

This means that, under the considered aggregators f and F, σ_I is the best mapping between S and S'. However, it will take TA $(\frac{n}{2})!$ iterations to discover σ_I. ∎

Theorem 3 Given a set of m algorithms for schema matching, and a pair of mutually commutative local and global aggregation functions $[\![f, F]\!]$, the MD algorithm correctly finds top-K

mappings with respect to the aggregative ranking in time $O(n^2m + C(\mathsf{TKM}))$, where $C(\mathsf{TKM})$ denotes the time complexity of the TKM algorithm.

Proof: The correctness is immediate by the definition of mutual commutativity. As F is assumed to be computable in time linear in the number of F's parameters, generating M^* takes time $O(n^2m)$. Thus, the overall complexity of MD is $O(n^2m + C(\mathsf{TKM}))$. For instance, using the TKM algorithm from [1], the time complexity of MD is $O(n^4K + n^2m)$. ∎

Theorem 5 Given a set of m algorithms for schema mappings, a pair of local and global aggregation functions $[\![f, F]\!]$, and a pair of mutually commutative functions $[\![h, H]\!]$ such that $[\![f, F]\!] \prec [\![h, H]\!]$, the MDB algorithm correctly finds top-K mappings with respect to $[\![f, F]\!]$.

Proof: Let Y be as in step 3 of MDB. We need only show that every mapping $\sigma \in Y$ has at least as high weight according to $[\![f, F]\!]$ as every mapping $\sigma' \notin Y$. By definition of Y, this is the case for each mapping $\sigma' \notin Y$ that has been seen by MDB. Thus, assume that σ' was not seen. However, by definitions of TKM and step 2b of MDB, for each such unseen σ' and for each $\sigma \in Y$ we have:

$$[\![f, F]\!](\sigma) \geq \tau \geq [\![h, H]\!](\sigma') \geq [\![f, F]\!](\sigma')$$

where τ is the value of τ_{MD} at termination of MDB. Thus, we have proven that Y contains top-K mappings with respect to $[\![f, F]\!]$. ∎

Theorem 6 Given a schema meta-matching problem instance, time complexity of TA on this instance can be exponentially worse than this of MDB.

Proof: The proof is by example of the corresponding problem instance: Consider the schema meta-matching problem exactly as in the proof of Theorem 2, and assume further that $x \in (0, 1]$. We already showed that TA on this problem instance performs $\Omega((\frac{n}{2})!)$ iterations. Recall that the aggregators f and F in this example stand for *product* and *min*, respectively. Hence, f and F are not mutually commutative, and thus MD cannot be used for this problem instance.

Now, consider a pair of functions $[\![h, H]\!]$, where both h and H stand for a simple *average*. Observe that, since the entries of both matrices $M^{(1)}$ and $M^{(2)}$ lie in the interval $[0, 1]$, we have $[\![f, F]\!] \prec [\![h, H]\!]$. Likewise, since h and H are (trivially) mutually commutative, we can solve this problem instance using MDB with $[\![h, H]\!]$.

The matrix M^*, constructed by MDB from the matrices $M^{(1)}$ and $M^{(2)}$ using H, is defined as below on the left, where on the right we illustrate such a matrix for $n = 4$:

$$M^*_{i,j} = \begin{cases} x/2, & (i \leq n/2) \wedge (j \leq n/2) \wedge (i \neq j) \\ x/2, & (i > n/2) \wedge (j > n/2) \wedge (i \neq j) \\ x - \epsilon, & i = j \\ 0, & \text{otherwise} \end{cases} \qquad M^* = \begin{pmatrix} x-\epsilon & x/2 & 0 & 0 \\ x/2 & x-\epsilon & 0 & 0 \\ 0 & 0 & x-\epsilon & x/2 \\ 0 & 0 & x/2 & x-\epsilon \end{pmatrix}$$

It is not hard to see that, for any $\epsilon < x/2$, the mapping σ processed in the *first* iteration of MDB will be the mapping σ_I, corresponding to the main diagonal of M^*. Likewise, in the proof of Theorem 2 we already showed that σ_I is the best mapping with respect to $[\![f, F]\!]$. Hence, the time complexity of TA on this problem instance with $K = 1$ is exponentially worse than this of MDB (with properly chosen upper bound $[\![h, H]\!]$). ∎

Theorem 7 Given a schema meta-matching problem instance, time complexity of MDB on this instance can be exponentially worse than this of TA.

Proof: Consider two algorithms, A_1 and A_2, and a pair of schemata S and S', each consisting of n concepts. Likewise, let the l-aggregator f be the *product* operator, and the g-aggregator F be the *min* operator.

Given S and S', the similarity matrices $M^{(1)}$ and $M^{(2)}$, induced by A_1 and A_2, respectively are as follows:

$$M^{(1)}_{i,j} = \begin{cases} \epsilon, & i = j \\ 0, & \text{otherwise} \end{cases} \qquad M^{(2)}_{i,j} = \begin{cases} 1 - 3\epsilon, & i = j \\ 1, & \text{otherwise} \end{cases}$$

for an arbitrary value of $\epsilon > 0$, such that $1 - 3\epsilon > 0$. Below we illustrate such matrices for $n = 4$:

$$M^{(1)} = \begin{pmatrix} \epsilon & 0 & 0 & 0 \\ 0 & \epsilon & 0 & 0 \\ 0 & 0 & \epsilon & 0 \\ 0 & 0 & 0 & \epsilon \end{pmatrix} \qquad M^{(2)} = \begin{pmatrix} 1-3\epsilon & 1 & 1 & 1 \\ 1 & 1-3\epsilon & 1 & 1 \\ 1 & 1 & 1-3\epsilon & 1 \\ 1 & 1 & 1 & 1-3\epsilon \end{pmatrix}$$

Considering the execution of TA on $M^{(1)}$ and $M^{(2)}$ as above, first notice that the only mapping σ, for which we have $f(\sigma, M^{(1)}) > 0$, is the mapping σ_I (i.e. the identity permutation). Therefore, σ_I will be discovered by TA right at the *first* iteration. Second, notice that all the entries of $M^{(1)}$ and $M^{(2)}$ lie in the interval $[0, 1]$. Thus, for all $\sigma_I \neq \sigma \in \Sigma$, we have $[\![f, F]\!](\sigma) = 0$. Finally, since $f(\sigma_I, M^{(2)}) > 0$, we have $[\![f, F]\!](\sigma) > 0$, and thus σ_I is the best mapping with respect to $[\![f, F]\!]$.

Clearly, the aggregators f and F are not mutually commutative. Now, consider a pair of functions $[\![h, H]\!]$, where both h and H stand for a simple *average*. It is worth noting that, since the entries of both matrices $M^{(1)}$ and $M^{(2)}$ lie in the interval $[0, 1]$, we have $[\![f, F]\!] \prec [\![h, H]\!]$. Likewise, since h and H are (trivially) mutually commutative, we can solve this problem instance using MDB with $[\![h, H]\!]$. The matrix M^*, constructed by MDB from the matrices $M^{(1)}$ and $M^{(2)}$ using H, is defined as below on the left, where on the right we illustrate such a matrix for $n = 4$:

$$M^*_{i,j} = \begin{cases} \frac{1}{2} - \epsilon, & i = j \\ \frac{1}{2}, & \text{otherwise} \end{cases} \qquad M^* = \begin{pmatrix} \frac{1}{2}-\epsilon & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2}-\epsilon & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2}-\epsilon & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2}-\epsilon \end{pmatrix}$$

For each mapping $\sigma \in \Sigma$, let k_σ be the number of concepts $i \in S$, such that $\sigma(i) = i$ (i.e. the number of the concept mappings in σ that lie on the main diagonal of M^*). For each $\sigma \in \Sigma$, we have $k_\sigma \in \{1, 2, \ldots, n-3, n-2, n\}$, and:

$$h(\sigma, M^*) = \begin{cases} \frac{1}{2}, & k_\sigma = 0, \\ \frac{1}{2} - \frac{k\epsilon}{n}, & 0 < k_\sigma \leq n-2, \\ \frac{1}{2} - \epsilon, & k_\sigma = n \end{cases}$$

Therefore, for each $\sigma_I \neq \sigma \in \Sigma$, we have $[\![h, H]\!](\sigma) > [\![h, H]\!](\sigma_I)$, and thus (the best mapping!) σ_I will be the *last* mapping discovered by MDB. ∎

Theorem 8 Given a set of m algorithms for schema mapping, a pair of local and global aggregation functions $[\![f, F]\!]$, and a pair of mutually commutative functions $[\![h, H]\!]$ such that $[\![f, F]\!] \prec [\![h, H]\!]$, the Hybrid algorithm correctly finds top-K mappings with respect to $[\![f, F]\!]$.

Proof: Let Y be the set of mappings as in step 3 of Hybrid. We need only show that every mapping $\sigma \in Y$ has at least as high weight according to $[\![f, F]\!]$ as every mapping $\sigma' \notin Y$. By definition of Y, this is the case for each mapping $\sigma' \notin Y$ that has been seen by Hybrid. Assume that σ' was not seen, and let τ', τ'_{TA}, and τ'_{MDB} be the value of τ, τ_{TA}, and τ_{MDB}, respectively, at termination of Hybrid.

If $\tau'_{MDB} > \tau_{TA}$, by monotonicity of F, we have $\tau' = \tau_{TA} \geq [\![f, F]\!](\sigma')$ for every $\sigma' \notin Y$. Otherwise, if $\tau'_{MDB} \leq \tau_{TA}$, by the definition of TKM, we have $\tau \geq [\![h, H]\!](\sigma') \geq [\![f, F]\!](\sigma')$ for every $\sigma' \notin Y$. But by definition of Y, for every $\sigma \in Y$ we have $[\![f, F]\!](\sigma) \geq \tau'$. Therefore, for every $\sigma' \notin Y$, we have $[\![f, F]\!](\sigma) \geq \tau' \geq [\![f, F]\!](\sigma')$, as desired. ∎